Recommendations for dCache, IFDH and SAM tools

Robert Illingworth, Dmitry Litvintsev, Marc Mengel, Andrew Norman, Gene Oleynik

Tuesday, July 28, 2015

The Scientific Computing Division was asked by Nova to have its data handling and storage experts make recommendations on the use of dCache, IFDH tools, SAM tools and in particular, NFS v4.1. We received a number of end user comments, went through them and have listed several recommendations below.

Keep in mind that dCache is not a file system, it is a large scale, high performance, storage and retrieval system, and does not have the semantics of a full file system. Some interfaces to dCache emulate a partial POSIX I/O interface, but are subject to the limitations of the underlying technology. As a result dCache cannot be used in the same manner as a traditional file system. This is elaborated on in sections following the recommendations.

Recommendations

1. We strongly recommend using SAM for cataloging files stored in dCache, and using fife_utils (and the underlying SAM and IFDH tools) to transfer files into, out of and within dCache. There are numerous reasons for this explained in the next section.
2. There is some concern that using NFS v4 file i/o on the tape backed dCache storage space could potentially result in performance degradation for the tape system, which would affect all experiments. The reasons for this concern are detailed in the next section. We recommend that NFS v4 ***not*** be used to perform file i/o in the tape-backed dCache storage spaces. We recommend using fife, SAM and IFDH tools, which can optimize staging files from tape into dCache, to manage and transfer files in this space.
3. While we recommend the use of fife SAM and IFDH tools for access to all of dCache, we see no reason not to use NFS v4 file i/o on *non tape-backed* persistent and scratch spaces.
4. dCache NFS v4 is best thought of not as a full file system, but rather another supported access protocol to dCache storage. When using NFS v4, remember the following dCache properties:
    a. Files are immutable. This includes their retention policies (scratch, persistent or tape backed).
    b. Performing a mv between areas with different types of retention policies doesn't change their retention policy, only the metadata. To change the retention policy a file must be copied to a space with the desired retention policy.
    c. Scratch and persistent areas are not tape-backed. If files are removed, they are not recoverable. For scratch, files have a limited life and will be automatically deleted to make room for new files.
5. BlueArc is not going away, in fact, we are purchasing additional BlueArc disk this quarter. BlueArc is a full file system that is suitable for interactive file use, for storing and compiling programs, etc. However, dCache is more suited in performance and capacity for analysis and production use. dCache can also be

used in conjunction with BlueArc. For example, if you have Grid jobs that want to read files that live on BlueArc; you can copy them to dCache scratch, and then give that location to your jobs to read from, and then stage output through dCache scratch into BlueArc disk.

6. If you are in doubt about whether some data needs to be kept for long-term archive, err on the side caution and copy it to tape backed read/write space.
7. There appears to a lack of awareness of the full set of capabilities that fife, SAM and IFDH tools provide. We recommend that man pages, with generous examples, be created for these tools.

About dCache, NFS and IFDH

There appears to be misunderstanding of what dCache is and how it can be used. We hope the brief description below helps clarify the role of dCache and set proper expectations of what it is designed to do well, and limitations to what it is not designed for or cannot do.

dCache was principally designed as a scalable front-end cache to tape storage systems. It is meant to provide low latency (compared to tape) reading of frequently accessed files using a variety of transfer protocols. The cache uses a Least Recently Used algorithm to evict infrequently accessed files in order to make room for new ones. This enables, on average, fast access to a much larger amount of data than can fit on the cache disk, greatly reducing the amount of disk required for production and analysis, and providing archival storage on tape.

This architecture has several consequences:

1. Since the authoritative copy of the file is on tape, files are immutable, that is, cannot be modified in place.
2. While latencies are generally low when the file is resident in the cache, when a file is not resident, latencies can be very high since the file needs to be read from tape, which consists of mounting the tape, seeking to the file on the tape, and transferring it to the cache. Time to get a file from tape is on the order of minutes at best.
3. Access latencies can be affected by how busy the system is. On a very busy system disk and tape resources can be tied up and requests can be internally queued for a long period of time.
4. Uncoordinated or random access of large numbers of files that are not resident in the cache can kill performance. dCache does not have knowledge of where files are located on tape, so cannot optimize the tape access. Worst case is a tape mount is required for each file that is requested. This can tie up drives mounting and dismounting tapes and killing performance. If this happens, large backlogs of requests can pile up resulting in a lengthy time to return back to normal performance.

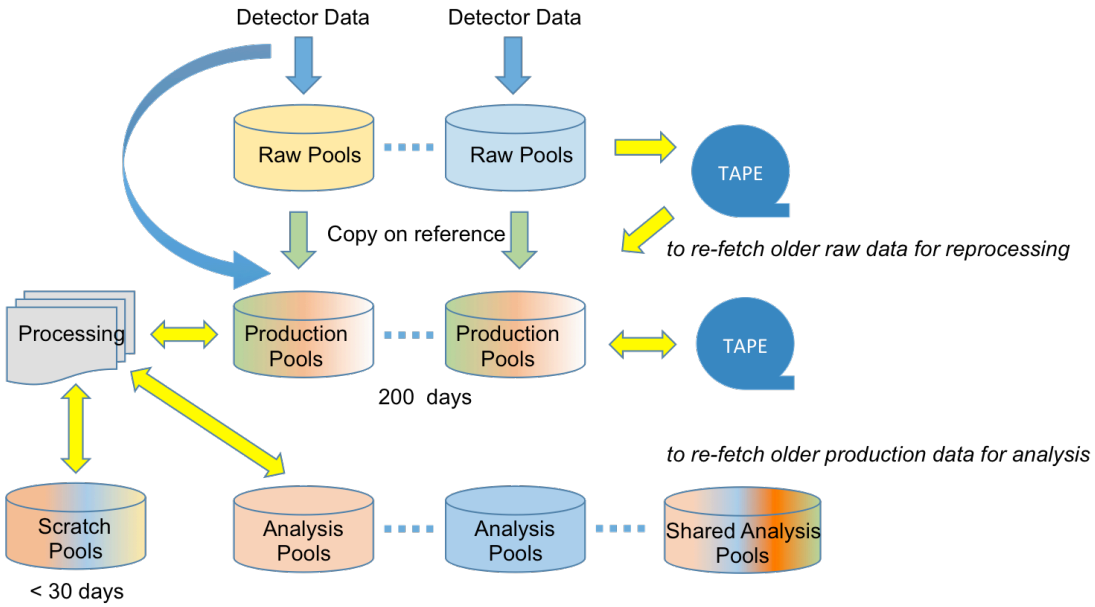
SAM and IFDH tools understand how to optimize access to datasets and provide file transfer tools and can greatly reduce the effects of the third and forth consequences in the above list.

dCache can also provide non-tape backed storage. This storage can act as a cache, with LRU eviction of files, or it can provide storage with no eviction policy. We refer to these types of dCache storage as "scratch" and "persistent" respectively (you may also hear these referred to as volatile and precious). Tape backed cache is referred to as "read/write" or "production".

A file written to persistent storage will not be automatically evicted. It will remain in dCache until removed by an end user. When persistent storage fills up, file writes will fail - users must manually remove files before more can be written.

The table and figure below indicates the properties of the various type of dCache storage. Included is a column "consequences" which is a list of the consequences in the numbered list above that apply, and target lifetimes.

Space	Retention policy	Tape backed	File lifetime	Consequences that apply
Read/write	LRU eviction	Y	< 200 days	1,2,3,4
Scratch	LRU eviction	N	< 30 days	1,3
Persistent	persistent	N	Until deleted	1,3



dCache access model (colors represent experiments. rainbow = shared)
dCache and the tape storage system at Fermilab, Enstore, share the Chimera/pnfs namespace. This hierarchical namespace contains metadata that describes the files: their name, location, whether or not they are in cache or on tape, and other information like file checksums. This namespace is not intended to be a catalog or an interface for file system, it contains metadata about files kept in the storage system. End users can use various protocols to access their files by specifying their Chimera filename.

The type of storage space that is written to is determined by (metadata) "tags" in the directory tree. We use the following convention when we setup pnfs for an experiment:

"/pNFS/usr/<experiment>/persistent" and below go to persistent space
"/pNFS/usr/<experiment>/scratch" and below go to scratch space
All other normally go to tape backed read/write space

The Chimera namespace can be presented to end users by NFS mounting a segment of the namespace to their host, so that users can conveniently access their files by name with the various dCache transfer protocols. dCache provides NFS v3 and, more recently, NFS v4 mounts.

With NFS v3 mounts, file i/o is not possible, only file metadata can be accessed and manipulated. With dCache NFS V4, file i/o is also possible, including a partial set of POSIX i/o. Even with the V4 file i/o capability, and a number of Linux tools can be used with it (like cp), dCache NFS does not have the semantics of a full file system. For one thing, files are immutable so cannot be edited in place. If the space accessed is tape backed, i/o can have latencies of minutes or longer when the file needs to be staged in from tape. Doing a mv of a file from scratch to persistent in the directory tree does not change the space to persistent – cp must be used to copy files from one type of storage space to another.

dCache NFS v4 is best thought of not as a full file system, but rather another supported access protocol to dCache storage with familiar semantics and tools. Many of the common Linux utilities can be used for NFS V4, but not all, and some can have unintended consequences when used with tape-backed dCache storage. NFS v4 mounted tape-backed shared read/write disk cache, while having the outward appearance of a normal file system, could lead users to perform i/o operations involving large numbers of files (for example rsync) that could unintentionally result in Enstore thrashing staging files from tape to disk, killing performance not only for them, but for all other experiments as well.

dCache does have the ability to restrict staging of files from tape to disk via NFS v4 to a specific user. This has an undesirable consequence however; for any other user, if the file is not already in the cache, they will not be able to access it – and their

request will time out. This will appear as random failures to the end user, which we do not think is acceptable.

For the above reasons, we recommend that IFDH tools be used to access files in general, and for the production shared read/write storage in particular. We recommend that SAM be used to register files in datasets that can be optimally manipulated (staged, cloned, etc), and to provide a catalog for the experiment's files. pNFS/Chimera namespace should not be used as a catalog for datasets.

For scratch or persistent space, we see no issues with using NFS v4 for i/o.

FIFE/IFDH/SAM tools recommendation

We looked at a number of IFDH complaints by end users. For example one was that a directory tree could not be recursively copied from scratch to persistent without writing a script. In fact, IFDH_cp does this (without cataloging) and THE SAM tools sam_add_dataset_followed by sam_clone_dataset can also do this and will register the files in the SAM catalog.

Based on this lack of awareness of the tools' capabilities, we recommend that IFDH and SAM tools should provide man pages with complete functionality and options. Also we think it may be valuable to create a video tutorial for new users.

A number of other end user issues were discussed. Recommendation or comments in ***[]***:

- IFDH tools (sometimes?) require certificates beyond Kerberos and it is confusing when a user does or does not need to request additional certificates. Can we make this more transparent. ***[need to have a kerberos principal and be in the right VO. IFDH tries to do this.]***
- Some tools are extremely verbose by default. One often gets hundreds of lines of feedback even for a successful copy command. Most of the statements are difficult to understand and some even give the impression that a successful copy fails. ***[debug on by default (IFDH_DEBUG set to 1 in their environment) normally nearly silent, but on failures lots of information that doesn't identify the problem is produced. Environment variable changed. Might make sense to instead have it off by default and overridden with a command line option that is quiet by default]***
- Users also report that sometimes commands fail but return a success code ***[please have users send a report or cut a service desk incident so these can be fixed].***
- I would say that even where the tools exist they're not polished enough, eg "IFDH cp", "IFDH mkdir" etc aren't drop in replacements for their unix equivalents. One example: "IFDH mkdir -p" doesn't work, you have to write a

recursive function for yourself ***[need a man page of what options are supported. Maybe some work to do here besides documenting, not clear. Man IFDH]***

- What is the proper way to move files between the various pNFS areas? ***[sam_clone_dataset tool to do this, for just several files IFDH_cp or if using NFS between scratch and persistent - cp]***
- Monitoring should be improved: If something fails how does a user know if dCache is having a problem, or if they did something incorrectly? ***[work in progress on this. dCache for instance could provide better information to IFDH monitoring rather than having IFDH scrape the dCache active transfer page and interpret the results]***
- What user activities should be avoided in pNFS: directories with large file multiplicity, ls –l… What else? ***[no strong recommendations here besides file count and restricting ls]***
- A big part of the push for NFS is due to users struggling with the pNFS tools and causing constant effort from other experts (experts who also struggle with these tools at times). How can we make the system more robust and easier to use for everyone? ***[we don't recommend using pNFS as a catalog. Files should be registered with SAM and use IFDH and SAM data tools]***

BlueArc vs. dCache space

Contrary to what you might have heard, BlueArc is not going away. It is suitable for home areas, development, compiling code, code, storage of small datasets for local analysis, and interactive use. BlueArc is not suitable for storage of large data sets for analysis.

dCache is very efficient at providing storage for high throughput computing, which is what it was designed for. It is not suitable for interactive use (immutable files), compiling, storing code, etc.

We recommend production data be stored on the dCache production read/write space with intermediary short term storage on the dCache scratch space, analysis on the dCache persistent or scratch space, and storage that requires interactive computing, compilation and small analysis on BlueArc.

We will try and define where the crossover for "small analysis" performance is by performing some measurements and reporting back (e.g. by running test case with chained ntuple on dCache, BlueArc).